Fault Tolerant Systems Design in VLSI Using Data Compression Under Constraints of Failure Probabilities – Overview and Status

SUNIL R. DAS

School of Information Technology and Engineering, Faculty of Engineering
University of Ottawa, Ottawa, Ontario K1N 6N5, CANADA
Department of Computer and Information Science, College of Arts and Sciences
Troy University, Montgomery, AL 36103, USA

Abstract: – The design of space-efficient support hardware for built-in self-testing (BIST) is of immense significance in the synthesis of present day very large-scale integration (VLSI) circuits and systems, particularly in the context of design paradigm shift from system-on-board to system-on-chip (SOC). This paper presents an overview of the general problem of designing zero-aliasing or aliasing-free space compression hardware in relation to embedded cores-based SOC for single stuck-line faults in particular, extending the well-known concepts of conventional switching theory, and of incompatibility relation to generate maximal compatibility classes (MCCs) utilizing graph theory concepts, based on optimal generalized sequence mergeability, as developed by the authors in earlier works. The paper briefly presents the mathematical basis of selection criteria for merger of an optimal number of outputs of the module under test (MUT) for realizing maximum compaction ratio in the design, along with extensive simulation results on ISCAS 85 combinational and ISCAS 89 full-scan sequential benchmark circuits, with simulation programs ATALANTA, FSIM, and COMPACTEST.

Key Words: – Aliasing-free space compactor, cores-based system-on-chip (SOC), maximal compatibility classes (MCCs), maximal minimally strongly connected (MMSC) subgraphs, undirected graphs and their cliques.

1 Introduction

VERY large-scale integration (VLSI) has added tremendous complexity to the test generation process of integrated circuits (ICs). With the unabated growth of the electronics industry, the integration densities and system complexities continue to increase, and thus the need for better and more efficient methods of testing of to guarantee reliable operations of chips, the mainstay of today's many sophisticated devices and products, is being constantly felt [1–18]. The very concept of testing has a relatively broad applicability, and finding the most effective testing techniques that can guarantee correct system performance is of immense practical significance. Generally, the price of testing integrated circuits (ICs) is rather prohibitive, accounting for 35% to 55% of their total manufacturing cost. Besides, testing a chip is also time-consuming, taking up to about one-half of the total design cycle time. The amount of time available for manufacturing, testing, and marketing a product, on the other hand, is on the decline. Moreover, as a result of diminishing trade barriers and global competition, customers now demand products of better quality at lower cost. In order to achieve this higher quality at lower cost, evidently the testing methods have to be improved. The conventional testing techniques of digital circuits require application of test patterns generated by a test pattern generator (TPG) to the module under test (MUT) and comparing the responses with known correct responses. For large circuits, because of higher storage requirements for the fault-free responses, the usual test procedures are sought to minimize the amount of needed storage [16].

Built-in self-testing (BIST) is a design process that provides the capability of solving many of the problems otherwise encountered in testing digital systems. It combines the concepts of both built-in test (BIT) and self-test (ST) in one termed built-in self-test (BIST). In BIST, test generation, test application, and response verification are all accomplished through built-in hardware, which allows different parts of a chip to be tested in parallel, reducing thereby the required testing time, besides eliminating the necessity for external test equipment. As the cost of testing is becoming the single major component of the manufacturing expense of a new product, BIST thus tends to reduce manufacturing and maintenance costs through improved diagnosis. Several companies such as Motorola, AT&T, IBM, and Intel have incorporated BIST in many of their products [3, 4, 6–8]. AT&T, for example, has incorporated BIST into more than 200 of their IC chips. The three large programmable logic arrays (PLAs) and microcode ROM in the Intel 80386 microprocessor were built-in self-tested [16–18]. The general-purpose microprocessor chip, Alpha AXP21164, and Motorola microprocessor 68020, were also tested using BIST techniques [4]. More recently, Intel, for its Pentium Pro architecture microprocessor, with its unique requirements of meeting very high production goals, superior performance standards, and impeccable test quality put strong emphasis on its design-for-test (DFT) direction [8]. A set of constraints, however, limits Intel's ability to tenaciously explore DFT and test generation techniques, *viz*. full-scan or partial-scan or scan-based BIST [2]. AMD's K6 processor is a reduced instruction set computer (RISC) core named enhanced RISC86 microarchitecture [7]. K6 processor incorporates BIST into its DFT process. Each RAM array of K6 processor has its own BIST controller. BIST executes simultaneously on all of the arrays for a predefined number of clock cycles that ensures completion for the largest array. Hence, BIST execution time depends on the size of the largest array [2]. AMD uses commercial automatic test pattern generation (ATPG) tool to create scan test patterns for stuck-faults in their processor. The DFT framework for 500-MHz IBM S/390 microprocessor utilizes a wide range of tests and techniques to guarantee superb reliability of components within a system [2]. Register arrays are tested through the scan chain, while nonregister memories are tested with programmable RAM BIST. Hewlett-Packard's PA8500 is a 0.25-μm superscalar processor that achieves fast but thorough test with its cache test hardware's ability to perform March tests, which is an effective way to detect several kinds of functional faults [6]. Digital's Alpha 21164 processor combines both structured and ad hoc DFT solutions, for which a combination of hardware and software BIST was adopted [2]. Sun Microsystems' UltraSparc processor incorporates several DFT constructs as well. The achievement of its quality performance coupled with reduced chip area conflicts with a design requirement that is easy to debug, test, and manufacture [2].

BIST is widely used to test embedded regular structures that exhibit a high degree of periodicity such as memory arrays (SRAMs, ROMs, FIFOs, and registers). A typical BIST environment uses a TPG that sends its outputs to an MUT, and output streams from the MUT are fed into a test data analyzer. A fault is detected if the test sequence is different from the response of the fault-free circuit. The test data analyzer is comprised of a response compaction unit (RCU), storage for the fault-free responses of the MUT, and a comparator. In order to reduce the amount of data represented by the fault-free and faulty MUT responses, data compression is used to create signatures (short binary sequences) from the MUT and its corresponding fault-free circuit. Signatures are compared and faults are detected if a match does not occur. BIST techniques may be used during normal functional operating conditions of the unit under test (on-line testing), as well as when a system is not carrying out its normal functions (off-line testing). In the case where detecting real-time errors is not that important, systems, boards, and chips can be tested in off-line BIST mode. BIST techniques use pseudoexhaustive or pseudorandom test patterns, or sometimes on-chip storing of reduced or compact test sets. Today, testing logic circuits exhaustively is seldom used, since only a few test vectors are needed to ensure full fault coverage for single stuck-line faults [16–18]. Reduced pattern test sets can be generated using existing algorithms such as FAN, and others [1, 2]. Built-in test generators can often generate such reduced test sets at low cost, making BIST techniques suitable for on-chip self-testing.

The subject paper focuses primarily on the response compaction process of BIST techniques that basically formulate into realizing appropriate means of reducing the test data volume coming from the MUT to a signature. The response compaction in BIST is carried out through a space compaction unit followed by time compaction. In general, P input sequences coming from an MUT are fed into a space compactor, providing L output streams of bits such that L << P; most often, test responses are compressed into only one sequence (L = 1). Space compaction brings a solution to the problem of achieving high-quality BIST of complex chips without the necessity of monitoring a large number of internal test points, reducing thereby testing time and area overhead by merging test sequences coming from these internal test points into a single stream of bits [11–13]. This single bit stream of length H is ultimately fed into a time compactor, and a shorter sequence of length B (B < H) is obtained at the output [9, 10]. The extra logic representing the compaction network, however, must be as simple as possible, to be easily embedded within the MUT, and should not introduce signal delays to affect either the test execution time or normal functionality of the module being tested. Moreover, the length of the signature must be as short as possible in order to minimize the amount of memory needed to store the fault-free response signatures. Also, signatures derived from faulty output responses and their corresponding fault-free signatures should not be the same, which unfortunately is not always the case. A fundamental problem with compaction techniques is error masking or aliasing [16–18] which occurs when the signatures from faulty output responses map into the fault-free signatures, usually calculated by identifying a good circuit, applying test patterns to it, and then having the compaction unit generate the fault-free references.

A major challenge in realizing efficient space compaction in BIST is the development of techniques that are simple, suitable for on-chip self-testing, require low area overhead, and have little adverse impact on the MUT performance. With this perspective in focus, this paper *revisits* the general problem of designing zero-aliasing BIST support hardware with applications targeted towards embedded cores-based system-on-chip (SOC) [15, 18], extending the well-known concepts of conventional switching theory, particularly those of cover table and frequency ordering commonly utilized in the simplification of switching functions, and of incompatibility relation as employed in the minimization of incomplete sequential machines, using graph theoretic concepts in the design [22–25], based on optimal generalized sequence mergeability as developed and applied by the authors in earlier works [14], for detectable single stuck-line faults of the MUT. This paper makes use of mathematically sound selection criteria of merger of an optimal number of output lines of the MUT to decide on the logic for zero-aliasing, achieving maximal compaction ratio in the process, as is evident from extensive simulation experiments conducted on ISCAS 85 combinational and ISCAS 89 full-scan sequential benchmark circuits.

2 Implementation of zero-aliasing space compression – mathematical basis

The mathematical basis [15] underlying the realization of proposed zero-aliasing space compaction is outlined in the following for the sake of clear understanding and completeness.

Property 1 Let A and B represent two of the outputs of an MUT. Let these MUT outputs be merged by a gate from the logic family AND/NAND, OR/NOR, and XOR/XNOR, and let the gate output be z_1. Then, we might envisage the under noted possible scenarios:

Case 1 Fault-free (FF) outputs = Faulty (F) outputs (outputs subject to the condition of MUT having faults), *viz*. FF = F $\Rightarrow$ Outputs A and B of the MUT do not detect any faults, and faults are hence not detectable at z_1.

Case 2 Only the faults that occur at A and B (subject to the condition of MUT having faults) are detectable at z_1 $\Rightarrow$ FF $\neq$ F.

Case 3 Faults occur at A and B but either all or some are not detectable at z_1 $\Rightarrow$ FF $\neq$ F. In this case, the faults missed at z_1 are detected additionally at other outputs of the MUT (besides A and B).

Definition 1 Let A, B, C, … be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, … be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT outputs when subjected to a compacted set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the two outputs A, B conforms to conditions of Cases 1-2 above (but not Case 3). If the MUT outputs A, B are now merged by an AND(NAND) gate, we define output lines A, B to be strongly AND(NAND) compatible, written as

(AB) s-AND(NAND) compatible.

Definition 2 Let A, B, C, … be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, … be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT outputs when subjected to a compacted set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the two outputs A, B conforms to conditions of Case 3 (but not Cases 1-2). If the MUT outputs A, B are now merged by an AND(NAND) gate, we define output lines A, B to be weakly AND(NAND) compatible, written as

(AB) w-AND(NAND) compatible.

Definition 3 Let A, B, C, … be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, … be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT outputs when subjected to a compacted set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the two outputs A, B conforms to none of the conditions as specified by Cases 1-3. If the MUT outputs A, B are now merged by an AND(NAND) gate, we define output lines A, B to be AND(NAND) incompatible, written as

(AB) AND(NAND) incompatible.

We can similarly define two lines (AB) being s-OR(NOR) compatible, w-OR(NOR) compatible, OR(NOR) incompatible, s-XOR(XNOR) compatible, w-XOR(XNOR) compatible, and XOR(XNOR) incompatible.

Definition 4 Let A, B, C, … be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, … be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to either one of the three conditions as specified by Cases 1-3, but unknown to us. If the MUT outputs A, B are merged under these conditions by an AND(NAND), OR(NOR), or XOR(XNOR) gate, then we define output lines A, B to be simply AND(NAND), OR(NOR), or XOR(XNOR) compatible, written as

(AB) AND(NAND), OR(NOR), or XOR(XNOR) compatible.

Theorem 1 Let A, B, C, … be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, … be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT outputs when subjected to a compacted set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-AND(NAND) compatible. Similarly, let the outputs B, C be s-AND(NAND) compatible, and the outputs A, C be s-AND(NAND) compatible. Then the outputs (ABC) are s-AND(NAND) compatible and all faults are detectable at z_1.

Theorem 2 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT outputs when subjected to a compacted set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs $A_1, A_2, \ldots, A_m$ conforms to conditions of Cases 1-2 above, so that the outputs $(A_1A_2\ldots A_m)$ are s-AND(NAND) compatible. Then, all faults are detectable at z_1.

Theorem 3 Let A, B, C, … be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs A, B, C, … be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT outputs when subjected to a compacted set of deterministic tests τ, $\tau \leq 2^n$, τ might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A, B conforms to conditions of Cases 1-2 above, so that the outputs A, B are s-OR(NOR) compatible. Similarly, let the outputs B, C be s-OR(NOR) compatible, and the outputs A, C be s-OR (NOR) compatible. Then the outputs (ABC) are s-OR(NOR) compatible and all faults are detectable at z_1.

Corollary 3.1 Let $A_1, A_2, \ldots, A_m$ be the different outputs of an n-input m-output MUT. Let the faults detected at the MUT outputs $A_1, A_2, \ldots, A_m$ be θ where $\theta \leq \beta$, the total number of detectable faults at the MUT output when subjected to a compact set of deterministic tests τ, $\tau \leq 2^n$, τ

might not be a minimal or nonminimal but complete set of tests, or to pseudorandom tests. Assume that the fault situation at the outputs A_1, A_2, … , A_m conforms to conditions of Case 3 above, so that the outputs A_1, A_2, … , A_m are w-AND(NAND) compatible. Then all faults may or may not be detected at z_1.

Again, identical conclusions can be derived if lines (ABC) are w-OR(NOR) compatible, w-XOR(XNOR) compatible, or a number of lines A_1, A_2, … , A_m are w-OR(NOR) compatible, w-XOR(XNOR) compatible.

In actual situations, we do not know (and also it is rather difficult to know) whether the merged outputs conform to conditions specified by Cases 1-3 as discussed, and as such we have to deal exclusively with the case of *simply* compatible. However, very recently, a novel approach to the solution of the problem utilizing the concept of fault grading [26] has been proposed, which renders the developed mathematical basis underlying the notion of strong and weak compatibilities really meaningful. But since this paper does not address the theory underlying that approach, it becomes necessary to check here every possible MUT output pair in a group for being *simply* compatible (AND/NAND, OR/NOR, or XOR/XNOR) to form a larger maximal or nonmaximal compatibility class.

3 Graph theoretic concepts and implementation of design approach

An important problem in relation to designing zero-aliasing space compression networks as proposed herein is to first find the sets of maximal compatibility classes (MCCs) of response data outputs of the MUT for logic families AND/NAND, OR/NOR, and XOR/XNOR, given the information of the corresponding pairs of incompatibles. In this paper, use has been made of available graph theoretic approaches in the solution of the problem. Some relevant basic concepts of graph theory as used in the paper in this regard might be relevant here for the sake of completeness [22, 24, 25].

A Approach based on generation of maximal complete subgraphs or cliques of undirected graphs using Bron-Kerbosch alogorithm

Some important basic definitions are given as follows.

Definition 5 An undirected graph A = (V, E). is defined as an ordered pair consisting of a finite set V of nodes or vertices, and a set of unordered pairs (v, w) of distinct vertices called edges. Any two vertices v, w in A are said to be adjacent to each other if $(v, w) \in E$. A set S of vertices of A is a complete subgraph if $(v, w) \in E$ for all pairs of distinct vertices $v, w \in S$. A maximal complete subgraph or clique of an undirected graph A is a complete subgraph that is not contained in any other complete subgraph of A. The complement of an undirected graph A = (V, E) is the graph $\bar{A} = (V, \bar{E})$, where $\bar{E} = \{(v, w) \mid v, w \in G, v \neq w, \text{ and } (v, w) \notin E\}$.

It is important to observe here that this clique detection problem of graph theory is identical to the problem of deriving the collection of maximal compatibility classes (MCCs) in a set of elements with compatibility relation. The maximal compatible problem as a counterpart of the clique problem has again been investigated by many authors in various disciplines. It is appropriate to remark here that the clique generation problem like some of the classical problems of combinatorics is an NP-complete problem [16], and as such is quite intractable.

Bron *et al.* [22] developed two backtracking algorithms for generating all cliques, using a branch-and-bound technique that cuts off branches that cannot lead to a clique. These algorithms were subsequently reported by Bron and Kerbosch and commonly known as Bron-Kerbosch algorithm in the literature [24]. Their first version is a straightforward implementation of the basic algorithm and generates cliques in a lexicographic order. The second version is derived from the first and generates cliques in an unpredictable order in an attempt to minimize the number of branches to be traversed. The authors implemented their algorithms with others. For the Moon-Moser graphs, the authors' second test case, the processing time for the first version was found proportional to 4^k, whereas for the second version of the algorithms, it was proportional to 3.14^k, for some constant k characteristic of the graphs. The algorithms need at most ½(M+3) storage locations to contain arrays of small integers, where M is the size of the largest connected component in the input graph. In our proposed approach for zero-aliasing space compaction, use has been made specifically of this well known Bron-Kerbosch algorithm for the generation of maximal compatibles (cliques) of response data outputs for logic families AND/NAND, OR/NOR, and XOR/XNOR, based on information of their pairs of incompatibles.

B Approach based on generation of maximal minimally strongly connected (MMSC) subgraphs – concepts

Definition 6 Consider an undirected graph A with n vertices, v_i, i = 1, 2, … , n. Two subgraphs A_a and A_b of A are said to be complementary to each other, if and only if, both A_a and A_b have the same set of vertices and one has edges connecting between those pairs of vertices that are not connected by edges in the other.

Definition 7 Consider a vertex v_i in an undirected graph A. The degree of v_i, $d(v_i)$, is the number of edges of A incident in v_i. The degree complement of a vertex v_i, $d'(v_i)$, is the degree of the vertex v_i in the complementary graph $\bar{A}$. Two vertices v_i and v_j in A are said to be minimally strongly connected, if and only if, v_i is reachable from v_j by a path of length 1. Otherwise, the vertices, if connected, are said to be nonminimally strongly connected. The degree complement of a nonminimally strongly connected pair of vertices (v_i, v_j) in A is written as $d'(v_i, v_j) = (k_1, k_2)$, where $d'(v_i) = k_1$, $d'(v_j) = k_2$.

Definition 8 A subgraph A_s of A is said to be minimally strongly connected (MSC), if and only if, every possible pair of vertices in A_s is minimally strongly connected. The subgraph A_s is said to be maximal minimally strongly connected (MMSC) if there does not exit any vertex outside of A_s which is minimally strongly connected with all the vertices of A_s.

Definition 9 Let (v_i, v_j) be a nonminimally strongly connected pair of vertices in A. Then splitting A into two subgraphs A_i and A_j such that A_i contains the vertex v_i and A_j contains the vertex v_j is to obtain two subgraphs A_i and A_j from A such that A_i contains all the vertices of A except v_j and A_j contains all the vertices of A except v_i, both A_i and A_j having all the existing edges of A connecting between relevant pairs of vertices. Obviously, $A_i \subseteq A$; $A_j \subseteq A$.

Definition 10 For any two distinct nonminimally strongly connected pairs of vertices (v_{i1}, v_{j1}) and (v_{i2}, v_{j2}) in A, let $d'(v_{i1}, v_{j1}) = (k_1, k_2)$ and $d'(v_{i2}, v_{j2}) = (r_1, r_2)$. If $k_1 + k_2 > r_1 + r_2$, then an ordering of the degree complements of the pairs of vertices can be made as $d'(v_{i1}, v_{j1}) \geq d'(v_{i2}, v_{j2})$, whereas, if $k_1 + k_2 < r_1 + r_2$, the ordering of the degree complements of the pairs of vertices can be made as $d'(v_{i2}, v_{j2}) \geq d'(v_{i1}, v_{j1})$. If, however, $k_1 + k_2 = r_1 + r_2$, the ordering can be made either as $d'(v_{i1}, v_{j1}) \geq d'(v_{i2}, v_{j2})$, or as $d'(v_{i2}, v_{j2}) \geq d'(v_{i1}, v_{j1})$. This kind of ordering ($\geq$) that can be established among degree complements of different nonminimally strongly connected pairs of vertices in an undirected graph is called the *magnitude ordering* of the degree complements of the pairs of vertices.

Theorem 4 Let A be an undirected graph, and let (v_i, v_j) be a nonminimally strongly connected pair of vertices in A. Let the graph A be split around (v_i, v_j) into two subgraphs A_i and A_j and let this process of splitting around nonminimally strongly connected pairs of vertices be iteratively applied to A_i and A_j and to all their subgraphs until in the resulting subgraphs there exist no more nonminimally strongly connected pairs of vertices. The final set of these subgraphs then includes all the MMSC subgraphs of A.

Theorem 5 Let A be an undirected graph, and let (v_i, v_j) be a nonminimally strongly connected pair of vertices of A having the highest degree complement in the magnitude ordering. If now the graph A is split around (v_i, v_j) into two subgraphs A_i and A_j, then in the resulting subgraphs the number of nonminimally strongly connected pairs of vertices will always be less than that when A will be split into subgraphs around any other nonminimally strongly connected pair having non-highest degree complement in the magnitude ordering.

Theorem 6 In the process of successively splitting an undirected graph A into subgraphs around nonminimally strongly connected pairs of vertices, let A_i and A_j be any two subgraphs obtained at different stages such that $A_i \subseteq A_j$, but A_i is not derived from A_j. Then, in finding only MMSC subgraphs, the subgraph A_i may be discarded in general.

4 Algorithms development

The developed zero-aliasing space compression approach consists of a set of algorithms: The first algorithm is for computing set of incompatible pairs [15] of response data outputs of the MUT for logic AND/NAND, OR/NOR/, and XOR/XNOR, while the second and third algorithms are for finding their maximal compatibility classes (MCCs) from the incompatible pairs based on the two different graph theoretic approaches as discussed. The final algorithm constructs the space compaction networks using the information of the generated maximal compatibility classes. All the different algorithms are presented below.

A Algorithm 1

This algorithm computes all incompatible pairs of the MUT output lines (pairs that do not produce 100% fault coverage) for logic AND/NAND, OR/NOR, and XOR/XNOR.

Step 1) Get the total number of output lines of the MUT.

Step 2) Generate all possible combinations $(v_i, v_{j)}$ of the MUT output lines, taking two at a time, and store all pairs of the output lines (v_i, v_j).

Step 3) Select the first pair from the list of all combined output lines (v_i, v_j).

Step 4) Merge the selected pair of output lines (v_i, v_j) using logic gates AND/NAND, OR/NOR, and XOR/XNOR, respectively, using only one type of logic gate at a time.

Step 5) Add a new output line to the original MUT corresponding to the outputs (v_i, v_j), one at a time.

Step 6) Discard the output lines (v_i, v_j) from the original MUT, and generate a new modified MUT.

Step 7) Inject stuck-at logic faults into the newly generated MUT and apply test patterns.

Step 8) If the fault coverage is less than 100%, then store the output pair (v_i, v_j) in the incompatible pairs database of logic AND/NAND, OR/NOR, and XOR/XNOR, respectively.

Step 9) Delete the pair just considered, from the list of all combined output lines (v_i, v_j), and select the next pair.

Step 10) Go to step 4 and continue until all pairs are exhausted.

B Algorithm 2

This algorithm is an implementation of the well-known graph theory technique of Bron and Kerbosch for computing all cliques in an undirected graph [22, 24]. We employ this as one graph theoretic approach for computing the MCCs of response data outputs of the MUT for logic families AND/NAND, OR/NOR, and XOR/XNOR. In the process, we use information of the incompatible pairs of the MUT output lines as generated by applying Algorithm 1 as given above. The algorithm is now described as follows.

Step 1) Calculate the total number of vertices in the undirected graph.

Step 2) Find the connected diagonal elements of the graph.

Step 3) Select a candidate point.

Step 4) Merge the selected candidate to a set called compsub, which is to be extended by a new point, or shrunk by a point on traveling along a branch of the backtracking tree.

Step 5) Generate a new set called candidates, which is the set of all points that will in due time serve as an extension to the present configuration of compsub.
Step 6) Create another set called not, which is the set of all points that, at an earlier stage, already served as an extension of the present configuration of compsub, and are now explicitly excluded.
Step 7) Remove all points not connected to the selected candidate, keeping the old sets intact.
Step 8) Call the extension operator to perform on the newly generated sets.
Step 9) Remove the selected candidate from the compsub, and add it to the old set not after returning.

C Algorithm 3

This algorithm also finds the MCCs from the same set of incompatible pairs of the MUT outputs as obtained by Algorithm 1 above, based on the implementation of the other graph theoretic approach as outlined previously [25]. The algorithm is provided below.
Step 1) From the undirected graph A (compatibility graph) representative of the incompatible pairs, find the magnitude ordering of degree complements of the nonminimally strongly connected (NMSC) pair of outputs of the MUT in A.
Step 2) Select an NMSC pair of outputs (v_i, v_j) in A, where (v_i, v_j) has the highest degree complement in the magnitude ordering. If more than one pair of outputs has the highest degree complement, select any one of these output pairs (v_i, v_j). Split A around (v_i, v_j) into two subgraphs A_i and A_j such that A_i contains all the outputs (vertices) of A except v_j and A_j contains all the outputs (vertices) of A except v_i.
a) Consider the subgraph A_i; check if there exists a subgraph A_k from which A_j is not derived, contains A_i. If so, discard the subgraph A_j; if not, take A_i and go to step 1.
b) Consider the subgraph A_j; check if there exists a subgraph A_m from which A_j is not derived, contains A_j. If so, discard the subgraph A_j; if not, take A_j and go to step 1.
Step 3) Follow steps 1 and 2 iteratively until in all the resulting subgraphs there does not exist any NMSC pair of outputs. The final set of subgraphs then includes all the MMSC subgraphs (MCCs) of A.
Step 4).In the set of subgraphs obtained after step 3, check if any subgraph is contained in another subgraph for possible cancellation of non-MMSC subgraphs. The resultant set, after cancellation, if any, gives all the MMSC subgraphs (MCCs) of A.

D Algorithm 4

This algorithm utilizes the knowledge of MCCs as obtained from either Algorithm 2 or Algorithm 3 to construct zero-aliasing space compactors for the MUT. The final algorithm is now given as follows.
Step 1) Define the possible maximum number of stages in the space compaction trees at the MUT output.
Step 2) Get the total number of output lines in the MUT. Continue the following steps until there is only a single output line (possibly).
Step 3) Find the sets of all MCCs from the MUT for logic AND/NAND, OR/NOR, and XOR/XNOR, employing Algorithm 2 or Algorithm 3.
Step 4) Select an MCC_i with large (possibly largest) number of output lines from the set of MCCs. Select the next large class during subsequent iteration, if 100% fault coverage is not achieved in the previous iteration from the same MUT.
Step 5) Merge selected output lines of the MCC_i using appropriate logic gates (AND/NAND, OR/NOR, or XOR/XNOR).
Step 6) Add a new output line corresponding to the selected merged outputs of MCC_i.
Step 7) Discard those MUT output lines that are already used in MCC_i.
Step 8) Search another MCC_j from the remaining output lines.
Step 9) Merge the selected output lines in MCC_j using appropriate logic gates.
Step 10) Add a new output line corresponding to the selected merged outputs of MCC_j.
Step 11) Discard the output lines that are already used in MCC_j.
Step 12) Go to step 8 as long as there are MCCs in the sets, and enough output lines.
Step 13) Find all the remaining output lines that do not belong to any of the selected MCCs.
Step 14) Merge all these remaining lines with XOR/XNOR gate.
Step 15) Add a new output line corresponding to these selected merged outputs.
Step 16) Inject stuck-at logic faults into the newly generated MUT (original MUT with COMPACTOR hardware).
Step 17) Compute fault coverage (FC) by applying input test patterns.
Step 18) If FC = 100%, then replace the old MUT with the new MUT and go to Step 2 for generating the next stage of the compactor.
Step 19) If FC < 100%, then merge all the remaining output lines with two-input XOR/XNOR gates, two output lines at a time.
Step 20) Add new output lines corresponding to the selected merged outputs.
Step 21) Inject stuck-at logic faults into the newly generated MUT (original MUT with COMPACTOR hardware).
Step 22) Compute FC by applying input test patterns.
Step 23) If FC < 100%, then continue to work on the same MUT. Go to step 4 for selecting a new MCC_k.
Step 24) If FC = 100%, then replace the old MUT with the new MUT, and go to step 2 for computing the next and subsequent stages of the compactor.

5 Experimental results

Extensive simulations runs were conducted to demonstrate the feasibility of the proposed zero-aliasing space compaction scheme using ISCAS 85 combinational benchmark circuits and ISCAS 89 full-scan sequential benchmark circuits. In our design experimentation, we used ATALANTA [19] (fault simulation program developed at the Virginia Polytechnic Institute and State University) as test generation tool to produce the fault-free output sequences needed to construct our space compactor circuits and to test the benchmark circuits using reduced test sets. We also used FSIM fault simulation program [20] that generates pseudorandom test sets, and COMPACTEST [21] program to generate the reduced test sets that detect *most* detectable single stuck-line faults for all the benchmark circuits. For each circuit, we determined the fault coverage without the compactor, fault coverage with the compactor, number of test vectors used to construct the compaction tree, simulation CPU time, number of test vectors applied, hardware overhead, and compaction ratio by running ATALANTA and FSIM programs on a SUN SPARC 5 workstation, and COMPACTEST program on IBM AIX machine.

The complete results on ISCAS 85 combinational and ISCAS 89 full-scan sequential benchmark circuits are listed in the multitude of tables that follow (Tables 1–8). The circuits with the compressors were tested with the same fault injection and test vectors for all the simulation programs FSIM, COMPACTEST, and ATALANTA. The fault coverage is considered 100%, if the faults detected at the MUT outputs and COMPACTOR outputs are the same, implying thereby that the COMPACTOR did not introduce any information loss. The results on simulation using HOPE were not provided due to space constraint.

Fig. 1 gives estimates of the hardware overhead for ISCAS 85 combinational benchmark circuits using ATALANTA simulation program. For estimating the hardware overhead, we used the ratio of the weighted gate count *metric*, *viz.* average fanins multiplied by the number of gates or gate count of the COMPACTOR and that of the total circuit comprised of the MUT and COMPACTOR. Fig. 2, on the other hand, gives compaction ratio for ISCAS 89 full-scan sequential benchmark circuits using ATALANTA.

6 Conclusions

This paper visits zero-aliasing space compaction problem of response data outputs of MUT with application specifically targeted towards digital embedded cores-based SOCs. The technique utilizes conventional switching theory concepts, *viz.* those of cover table, frequency ordering, and compatibility relation together with those of strong and weak compatibilities of response data outputs, in the selection of specific gates for merger of an arbitrary but optimal number of output bit streams from the MUT. The techniques, illustrated with details of design of space compactors for ISCAS 85 combinational and ISCAS 89 full-scan sequential benchmark circuits with ATALANTA, FSIM, and COMPACTEST simulation programs, confirm the usefulness of the suggested approach, its simplicity, resulting low area overhead, and full fault coverage for single stuck-line faults, making it suitable in a VLSI design environment as BIST support hardware. In the sequel, it is evident from the experimental results that the suggested approach, though relies on restricted use of heuristics, still could be considered simple and robust enough in its design methodology for single stuck-line faults of the MUT. With advances in computational resources, evidently this heuristic space compaction algorithm might be improved upon for better efficiency in respect of time and storage.

References:

[1] Rajsuman, R.: 'System-on-a-chip: design and test' (Artech House, Boston, MA, 2000).

[2] Mourad, S., and Zorian, Y.: 'Principles of testing electronic systems' (Wiley, New York, 2000).

[3] Kuban, J. R., and Bruce, W. C.: 'Self-testing the Motorola MC6804P2', *IEEE Des. Test Comput.*, Vol. 1, 1984, pp. 33-41.

[4] Daniels, R. G., and Bruce, W. B.: 'Built-in self-test trends in Motorola microprocessors', *IEEE Des. Test Comput.*, Vol. 2, 1985, pp. 64-71.

[5] Das, S. R.: 'Built-in self-testing of VLSI circuits', *IEEE Potentials*, Vol. 10, 1991, pp. 23-26.

[6] Brauch, J., and Fleischman, J.: 'Design of cache test hardware on the HP PA8500', *IEEE Des. Test Comput.*, Vol. 15, 1998, pp. 58-63.

[7] Fetherston, R. S., Shaik, I. P., and Ma, S. C.: 'Testability features of the AMD-K6 microprocessor', *IEEE Des. Test Comput.*, Vol. 15, 1998, pp. 64-69.

[8] Carbine, A.: 'Pentium pro processor design for test and debug', *IEEE Des. Test Comput.*, Vol. 15, 1998, pp. 77-82.

[9] Frohwerk, R. A.: 'Signature analysis – a new digital field service method', *Hewlett-Packard J.*, Vol. 28, 1977, pp. 2-8.

[10] Jone, W. -B., and Das, S. R.: 'Multiple-output parity bit signature for exhaustive testing', *J. Electron. Tes., Theory Appl.*, Vol. 1, 1990, pp. 175-178.

[11] Li, Y. K., and Robinson, J. P.: 'Space compression method with output data modification', *IEEE Trans. Comput.-Aided Des.*, Vol. 6, 1987, pp. 290-294.

[12] Jone, W. –B., and Das, S. R.: 'Space compression method for built-in self-testing of VLSI circuits', *Int. J. Comput. Aided VLSI Des.*, Vol. 3, 1991, pp. 309-322.

[13] Pouya, B., and Touba, N. A.: 'Synthesis of zero-aliasing elementary-tree space compactors'. Proc. VLSI Test Symp., 1998, pp. 70-77.

[14] Das, S. R., Barakat, T. F., Petriu, E. M., Assaf, M. H., and Chakrabarty, K.: 'Space compression revisited', *IEEE Trans. Instrum. Meas.*, Vol. 49, 2000, pp. 690-705.

[15] Das, S. R., Assaf, M. H., Petriu, E. M., Jone, W. –B., and Chakrabarty, K.: 'A novel approach to designing aliasing-free space compactors based on switching theory formulation'. Proc. IEEE Instrum. Meas.Tech. Conf., 2001, vol. 1, pp. 198-203.

[16] Das, S. R., Ramamoorthy, C. V., Assaf, M. H., Petriu, E. M., and Jone, W. –B.: 'Fault tolerance in systems design in VLSI using data compression under constraints of failure probabilities', *IEEE Trans. Instrum. Meas.*, Vol. 50, 2001, pp. 1725-1747.

[17] Das, S. R.: 'Getting errors to catch themselves – self-testing of VLSI circuits with built-in hardware', *IEEE Trans. Instrum. Meas.*, Vol. 54, 2005, pp. 941-955.
[18] Das, S. R.: 'Self-testing of cores-based embedded systems with built-in hardware', *Proc. IEE – Circuits Devices Syst.*, Vol. 152, 2005, pp. 539-546.
[19] Lee, H. K., and Ha, D. S.: 'On the generation of test patterns for combinational circuits'. Tech. Rep. 12-93; Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1993.
[20] Lee, H. K., and Ha, D. S.: 'An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation'. Proc. Int. Test Conf., 1991, pp. 946-955.
[21] Pomeranz, I., Reddy, L. N., and Reddy, S. M.: 'COMPACTEST: a method to generate compact test sets for combinational circuits'. Proc. Int. Test Conf., 1991, pp. 194-203.
[22] Bron, C., Kerbosch, J., and Schell, H. J.: 'Finding cliques in an undirected graph'. Tech. Rep. BKS-1; Technical University of Eindhoven, Eindhoven, The Netherlands, 1972.
[23] Das, S. R.: 'On a new approach for finding all the modified cut-sets in an incompatibility graph', *IEEE Trans. Comput.*, Vol. 22, 1973, pp. 187-193.
[24] Bron, C., and Kerbosch, J.: 'Finding all cliques of an undirected graph', *Commun. ACM*, Vol. 16, 1973, pp. 575-577.
[25] Das, S. R., Sheng, C. L., Chen, Z., and Lin, T.: 'Magnitude ordering of degree complements of certain node pairs in an undirected graph and an algorithm to find a class of maximal subgraphs', *Int. J. Comput. Elec. Eng.*, Vol. 6, 1979, pp. 139-151.
[26] Das, S. R., Mukherjee, S., Petriu, E. M., Assaf, M. H., and Hossain, A.: 'Space compaction for embedded cores-based system-on-chips using fault graded output merger'. Proc. IEEE Instrum. Meas. Tech. Conf., 2007, pp. 1-5.

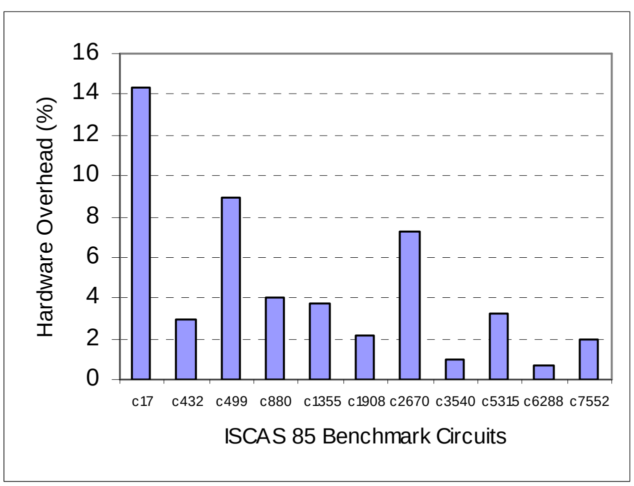

Fig. 1 Area overhead of compaction networks for ISCAS 85 combinational benchmark circuits using ATALANTA.

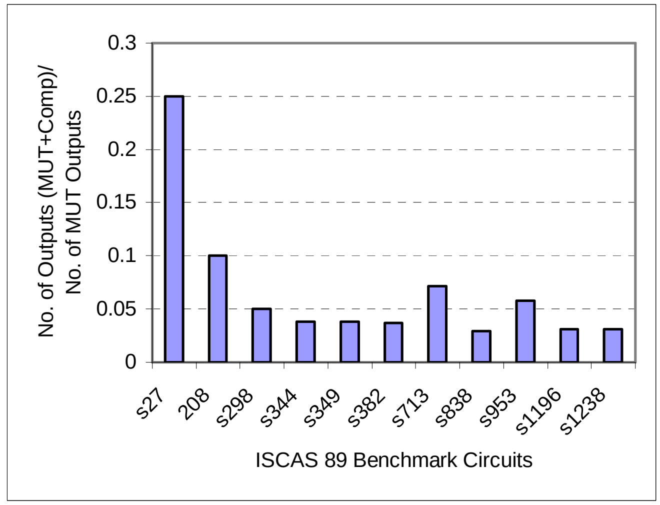

Fig. 2 Compaction ratio for ISCAS 89 full-scan sequential benchmark circuits using ATALANTA.

Table 1 Simulation results of ISCAS 85 combinational benchmark circuits using ATALANTA without space compactors

Circuit Name	Applied Test Vectors	No. of Faults Injected	No. of Outputs	Fault Coverage (%)
c17	7	22	2	100.00
c432	76	520	7	100.00
c499	66	750	32	100.00
c880	107	942	26	100.00
c1355	105	1566	32	100.00
c1908	184	1870	25	100.00
c2670	182	2630	140	100.00
c3540	253	3291	22	100.00
c5315	197	5291	123	100.00
c6288	53	7710	32	100.00
c7552	376	7419	108	100.00

Table 2 Simulation results of ISCAS 85 combinational benchmark circuits using FSIM without space compactors

Circuit Name	Applied Test Vectors	No. of Faults Injected	No. of Outputs	Fault Coverage (%)
c17	32	22	2	100.00
c432	544	520	7	100.00
c499	1312	750	32	100.00
c880	5480	942	26	100.00
c1355	2124	1566	32	100.00
c1908	29472	1870	25	100.00
c2670	6378144	2630	140	100.00
c3540	38848	3291	22	100.00
c5315	4576	5291	123	100.00
c6288	128	7710	32	100.00
c7552	10000000	7419	108	99.407

Table 3 Simulation results of ISCAS 85 combinational benchmark circuits using COMPACTEST without space compactors

Circuit Name	Applied Test Vectors	CPU Simulation Time (Secs)	No. of Outputs	Fault Coverage (%)
c17	4	0.01	2	100.00
c432	44	5.09	7	99.430
c499	65	8.36	32	98.990
c880	30	1.85	26	100.00
c1355	96	2.54	32	99.480
c1908	137	13.39	25	99.230
c2670	68	96.78	140	95.520
c3540	110	278.45	22	95.920
c5315	55	35.74	123	98.890
c6288	16	68.16	32	99.330
c7552	85	164.23	108	98.440

Table 4 Simulation results of ISCAS 85 combinational benchmark circuits using ATALANTA with space compactors

Circuit Name	Applied Test Vectors	No. of Faults Injected	No. of Outputs	Fault Coverage (%)
c17	10	22	1	100.00
c432	124	520	1	100.00
c499	169	750	1	100.00
c880	223	940	1	100.00
c1355	220	1566	1	100.00
c1908	313	1870	1	100.00
c2670	496	2630	3	100.00
c3540	270	3291	1	100.00
c5315	692	5291	1	100.00
c6288	65	7710	1	100.00

Table 5 Simulation results of ISCAS 85 combinational benchmark circuits using FSIM with space compactors

Circuit Name	Applied Test Vectors	No. of Faults Injected	No. of Outputs	Fault Coverage (%)
c17	45	22	1	100.00
c432	2752	520	1	100.00
c499	10929363	750	1	100.00
c880	97055712	940	1	100.00
c1355	100000000	1566	1	94.994
c1908	96283712	1870	1	100.00
c2670	100000000	2630	3	98.869
c3540	301824	3291	1	100.00
c5315	1316134912	5291	1	100.00
c6288	224	7710	1	100.00

Table 6 Simulation results of ISCAS 85 combinational benchmark circuits using FSIM with space compactors tested with compacted test vectors

Circuit Name	Applied Test Vectors	No. of Faults Injected	No. of Outputs	Fault Coverage (%)
c17	7	22	1	100.00
c432	80	520	1	100.00
c499	100	750	1	100.00
c880	159	940	1	100.00
c1355	124	1566	1	100.00
c1908	199	1870	1	100.00
c2670	366	2630	3	100.00
c3540	263	3291	1	100.00
c5315	686	5291	1	100.00
c6288	63	7710	1	100.00

Table 7 Simulation results of ISCAS 89 full-scan sequential benchmark circuits using ATALANTA/FSIM with space compactors

Circuit Name	No. of Faults Injected	Fault Coverage (without Compactor) (%)	No. of Outputs (after Compaction)	Fault Coverage (with Compactor) (%)
s27	32	100.00	1	100.00
s208	214	100.00	1	100.00
s298	306	100.00	1	100.00
s344	340	100.00	1	100.00
s349	348	100.00	1	100.00
s382	397	100.00	1	100.00
s713	921	100.00	3	100.00
s838	187	100.00	1	100.00
s953	81	100.00	3	100.00
s1196	1025	100.00	1	100.00
s1238	1035	100.00	1	100.00

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant A 4750.